Automatic Phonetic Transcription of Words Based On Sparse Data

Pages 61 to 70 of W. Daelemans, A. van den Bosch, and A. Weijters (Editors), Workshop Notes of the ECML/MLnet Workshop on Empirical Learning of Natural Language Processing Tasks, April 26, 1997, Prague, Czech Republic

Automatic Phonetic Transcription of Words Based On Sparse Data

Maria Wolters[(i)] and Antal van den Bosch[(ii)]

[(i)] Institut für Kommunikationsforschung und Phonetik, Universität Bonn
Poppelsdorfer Allee 47, 53113 Bonn, Germany
mwo@asl1.ikp.uni-bonn.de

[(ii)] Department of Computer Science, Universiteit Maastricht
PO Box 616, 6200 MD Maastricht, The Netherlands
antal@cs.unimaas.nl

Abstract

The relation between the orthography and the phonology of a language has traditionally been modelled by hand–crafted rule sets. Machine-learning (ML) approaches offer a means to gather this knowledge automatically. Problems arise when the training material is sparse. Generalising from sparse data is a well-known problem for many ML algorithms. We present experiments in which connectionist, instance–based, and decision–tree learning algorithms are applied to a small corpus of Scottish Gaelic. instance-based learning in the IB1-IG algorithm yields the best generalisation performance, and that most algorithms tested perform tolerably well. Given the availability of a lexicon, even if it is sparse, ML is a valuable and efficient tool for automatic phonetic transcription of written text.

1 The Problem

Experienced readers can read text aloud fluently and without pronunciation errors. But can we simulate this performance on a computer? This question is especially relevant for text–to–speech (TTS) synthesis. In a TTS system, orthographic text first has to be converted into a sequence of orthophones, which describe the pronunciation norm. This phonetic transcription is the main input of the synthesis module[1].

[1]Further processing steps are not considered here; for an overview, see (Allen et al., 1987).

The classic approach to automatic phonetic transcription (APT) is a large lexicon supplemented with a hand-crafted rule set. Many researchers have tried to replace rule sets using machine learning (ML) algorithms trained on the lexicon, but with mixed success. The performance of most algorithms still falls far below the mark of 80–90% correct words which is needed in high–quality text–to–speech synthesis (Yvon, 1996). However, Bakiri and Dietterich (1993) have shown that their approach based on ID-3 (Quinlan, 1986) decision trees outperforms the sophisticated DECTalk rule set for English (Allen et al., 1987); (Van den Bosch and Daelemans, 1993; Daelemans and Van den Bosch, 1997) report similar results for Dutch. In both cases, the training corpora contained around 18000, and the test corpora around 2000 words.

With the exception of (Dietterich and Bakiri, 1995), most researchers have relied on large machine readable pronunciation dictionaries for training and test data. However, for most languages, the necessary corpora have to be gathered and typed in first, because modern standard pronunciation dictionaries are available neither on paper nor in machine-readable form. While producing a large, well–debugged corpus takes longer than hand–crafting a rule set, a small corpus of about 1000–2000 words can be gathered in 1–2 weeks. Therefore, using ML algorithms is only worthwhile if they produce good results with little data.

In this paper, we examine the performance of ML algorithms on a Scottish Gaelic corpus of 1000 words. Section 2 provides a brief overview of the algorithms tested and explains why they were chosen. In section 3, we compare the performance of these algorithms on the Gaelic corpus. Section 4 presents some preliminary conclusions.

2 Choice of Algorithms

Two types of ML approaches to APT can be found in the literature:

chunk–based : A sequence of letters is mapped onto a *sequence of phonemes.*

phoneme–based : A sequence of letters is mapped onto a *phoneme.*

Although chunk–based approaches are psycholinguistically plausible (cf. Glushko, 1979), they are not suitable for minority-language APT. Algorithms in the tradition of PRONOUNCE (Dedina and Nusbaum, 1991) rely on extensive statistics about letter/phone correspondences which cannot be estimated adequately from tiny corpora. JUPA (Yvon, 1996), which recombines dictionary entries, does not produce any output for 30–40% of the test words when trained on 2000 words only, and similar problems should occur with the more sophisticated algorithms Yvon describes.

Therefore, we have to rely on phoneme–based approaches. Usually, a window of $2n + 1$ characters is shifted across the input word. The nth character of this win-

dow is transcribed, the other $2n$ serve as context. Because of the limited window length, it is difficult to capture morphophonological alternations like English Trisyllabic Shortening as in `divine` – `divinity`, and stress shifts as in `photograph` – `photography`.

Three types of phoneme-based approaches have yielded good results for large corpora: neural networks (Sejnowski and Rosenberg, 1987), decision trees (Dietterich et al., 1995), and instance–based learning (Van den Bosch and Daelemans, 1993).

2.1 Neural networks

Artificial neural networks (ANN) consist of simple processing units with weighted connections. The units are usually grouped into an input layer, an output layer, and one or more hidden layers. The best results on APT so far have been achieved using a simple feed-forward topology[2] and Backpropagation with Momentum (Rumelhart et al., 1986).

The ANN approach tested here was proposed in (Wolters, 1996). First, a feed–forward ANN is trained using Backpropagation with Momentum until the error on a validation set starts to rise (*early stopping*). This way, we avoid overfitting of the training data, which results in bad generalisation performance for neural networks. Usually, we find that the shorter the number of training epochs, the less precise the adjustment of the weights and the more noisy the internal distributed representations. To reduce this noise as much as possible, the net output is classified again. For this second stage, we use Learning Vector Quantization (LVQ, (Kohonen et al., 1996)). LVQ computes a set of no_{cod} codebook vectors which describe no_{class} classes (here: orthophones). An instance is classified by determining the classes of the k most similar codebook vectors and associating it with the most frequent class (*k–nearest–neighbour* classification).

2.2 Instance–Based Learning

Like LVQ, instance–based learning (IBL) descends from the k-nearest neighbour algorithm (Devijver and Kittler, 1982; Aha et al., 1991). In IBL, the basis for classification is not a set of codebook vectors, but a set of *exemplars*, instances encountered earlier in classification. IBL is a form of *lazy learning*, where learning only involves storing instances in memory, while computational effort is put into classification. On the contrary, in *eager learning*, computational effort is put mainly into learning. ANN and decision trees are eager algorithms. simple and robust approaches within the group of Case–Based Reasoning algorithms (CBR) (Kolodner, 1993), because it is based on feature-value vectors rather than on more complex expressions such as those in first-order logic (Kolodner, 1993; Lavrač and Džeroski, 1994).

[2] *feedforward*: the output of the units in layer i is only fed to units in layer $j > i$.

We examine two IBL algorithms, viz. IB1 and IB1-IG. IB1 (Aha et al., 1991; Daelemans et al., 1997) constructs a database of instances during learning. An instance consists of a fixed-length vector of n feature-value pairs, and an information field containing its class(es). When the classification of a feature-value vector is ambiguous, the frequencies of the relevant classes in the training material are calculated and the frequency information is stored together with the instance in the instance base. New instances X are classified by matching them to all instances Y in the instance base, calculating the distance $\Delta(X,Y)$ between X and each of the Ys using the distance function given in Eq. 1:

$$\Delta(X,Y) = \sum_{i=1}^{n} W(f_i)\delta(x_i,y_i) \quad (1)$$

where $W(fi)$ is the weight of the ith feature, and $\delta(x_i,y_i)$ is the distance between the values of the ith feature in instances X and Y. When feature values are symbolic, as with our data, $\delta(x_i,y_i) = 0$ when $x_i = y_i$, and $\delta(x_i,y_i) = 1$ when $x_i \neq y_i$.

IB1-IG (Daelemans and Van den Bosch, 1992) differs from IB1 in the weighting function $W(f_i)$ (cf. Eq. 1). The weighting function of IB1-IG, $W'(f_i)$, represents the *information gain* (Quinlan, 1993) of feature f_i. The information gain of a feature expresses the relevance of a feature for classification relative to the other features. in the distance function (Eq. 1), instances that match on features with a relatively high information gain are regarded as less distant (more alike) than instances that match on features with a lower information gain.

2.3 Decision Trees

Top-down induction of decision trees (TDIDT) is a well-developed field within artificial intelligence[3]. (TDIDT) is based on the assumption that the similarity information stored in an exemplar base can be compressed in a tree without significantly affecting generalisation. Learning in TDIDT is eager since decision trees are constructed during learning; classification effort is low since it involves non-backtracking deterministic traversal through the induced tree. Two decision tree algorithms are evaluated here: IGTREE (information-gain tree, (Daelemans et al., 1997)) and SCT (semantic classification trees, (Kuhn and De Mori, 1995)).

IGTREE (Daelemans et al., 1997) was designed as an optimised approximation of IB1-IG. In IGTREE, information gain is used as a guiding function to *compress* the instance base into a decision tree. Nodes are connected via arcs denoting feature values. Information gain is used in IGTREE to determine the order in which feature values are added as arcs to the tree. An instance is stored in the tree as a path of arcs whose terminal node (=*leaf*) specifies its class. When storing feature-value information, arcs representing the values of the feature with the highest information gain are created first, then arcs for the values of the feature with the second-highest

[3]see e.g. (Quinlan, 1993) for an overview

information gain, etc., until the classification information represented by a path is unambiguous. Short paths in the tree represent instances with relatively regular classifications, whereas long paths represent instances with irregular, exceptional, or noisy classifications.

Apart from storing uniquely identified class labels at each leaf, IGTREE stores information on the *default* classification at each non-terminal node. This default is the most frequent classification of those instances which are covered by the subtree below that node. A new instance is classified by matching its feature values with the arcs in the order of the overall feature information gain. When a leaf is reached, the instance is assigned the class stored at the leaf; otherwise, it is assigned the default classification associated with the last matching non-terminal node.

Semantic Classification Trees (SCT) were introduced by (Kuhn and De Mori, 1995) for Natural Language Understanding and have been applied successfully to the classification of dialogue acts by keyword spotting (Mast et al., 1995). In SCTs, the class of an instance is determined by matching it against a set of regular expressions. At each node, only one regular expression is tested. There are two branches, one for "match" and one for "no match". While tests are stored at nodes, classes are stored at leaves. To avoid overgeneralisation, the trees are trained using the algorithm of (Gelfand et al., 1991).

In contrast to neural nets, SCTs cannot extract equivalence classes of attributes from the data such as the class of vowel graphemes. However, the algorithm does not need any windowing; it can access the complete word quite efficiently by adequate regular expressions.

3 Comparison of Algorithm Performance

3.1 The Data Set

The algorithms were tested on a dictionary of 1003 phonetically transcribed Scottish Gaelic words (Wolters, 1997). The transcriptions reflect the Gaelic of Point, Isle of Lewis, Outer Hebrides. Scottish Gaelic is a minority language with about 80,000 speakers. Its orthography is rather complex. It was codified in the 18th century, and the dialect on which it is based has nearly died out today. The corpus is hand–aligned and contains both zero graphemes and zero phonemes[4]. The transcriptions are largely allophonic; 104 allophone classes are used. A window length of 7 yields on average 64 patterns per class. which often cover several different grapheme-phone

[4]Introducing zero elements eliminates the problem of parsing a sequence of letters into graphemes, functional units that correspond to a phoneme. It is basically a preprocessing task on the data level. In Gaelic, zero graphemes are necessary e.g. for preaffricated plosives, where we have correspondences like k → /xk/. Because the rules for inserting zero graphemes are very regular, their presence should not distort classification results significantly.

correspondences. On average, 3,78% of all training instances (1.57% of all types) are ambiguous, but less than 1% of all test instances. Vowel graphemes are especially susceptible to errors, since they are also used to encode consonant quality[5].

3.2 Method

All algorithms were trained using 10-fold cross validation (Weiss and Kulikowski, 1991) to allow for significance tests on performance differences.

The ANN consists of 1 input layer of 7×5 units, 2 hidden layers of 100 units each, and 1 output layer of 22 units. The size of the hidden layers was motivated by two main considerations. First, a large number of connections means a large variance with the potential to accomodate very complex hidden representations, and secondly, a size of 100–200 hidden units is quite common for this problem in the psycholinguistic/speech processing literature.

Letters were encoded using a binary code, phones using phonological features (Halle, 1992). For sparse data, it is advisable to keep the dimensions of input and output space small, because it is harder to estimate a function of many variables (i.e., high dimensional output space) on the basis of sparse data than it is to estimate a function of few variables (i.e., low dimensional output space)[6]. Since there was not enough data for a separate validation set, the test set was used for early stopping. The LVQ–codebook consisted of 2000, roughly 1/3 of the total number of patterns.

The SCT input was not coded using the window technique, because it accepts input of variable length and does not explicitly demand that features be in a certain order. Instead, each instance consisted of the source word and the position of the phoneme to be transcribed. This way, SCT disposes of all relevant information except for part-of-speech and semantic information needed for resolving word–level ambiguities.

3.3 Results

On the training set, we obtain near perfect recall for IB1, IB1-IG, and IGTREE (c.f. Fig. 1). The small remaining error is mostly due to ambiguity in the data. Recall is slightly worse for ANN, and significantly worse for SCT.

On the test set, however, the picture changes slightly, as can be seen in Fig. 2. Here, ANN–LVQ, IB1-IG and IGTREE provide the best generalisation performance, with IB1-IG significantly better than the other two algorithms ($p < 0.05$). Furthermore, weighting the contribution of the letters improves the generalisation performance of IBL significantly ($p < 0.001$). Why this superiority of the nearest neighbour classifier

[5] For example, in *cait* ("the cats"), `i` only serves as a cue to the palatality of /t/.

[6] see also the experiments reported in (Wolters, 1997) on the Gaelic corpus with different input and output representations.

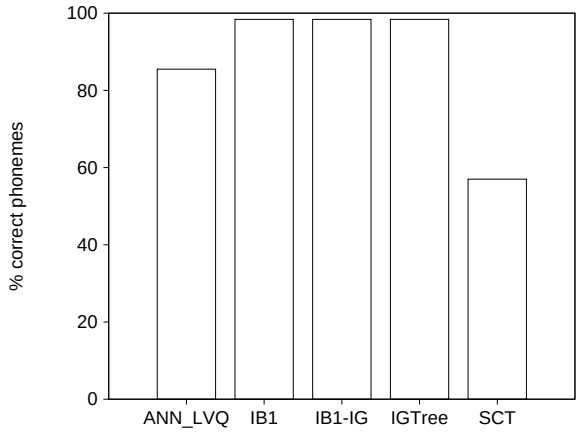

Figure 1: Average reproduction accuracy on training set in percentage of correctly classified phones

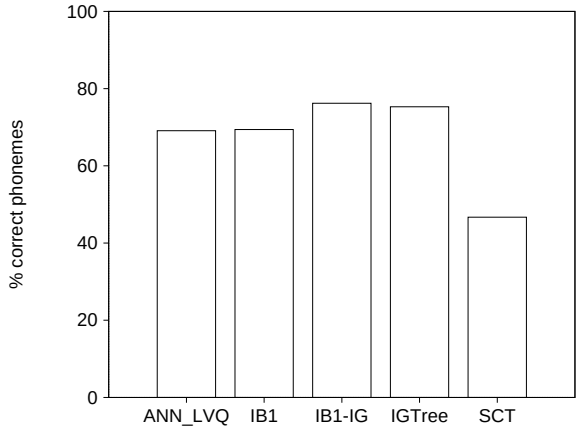

Figure 2: Average generalization accuracy on test set in percentage of correctly classified phones

IB1-IG? Three aspects of learning in IB1-IG are advantageous in lgeneralising from sparse data:

- *storing all training examples.* Many patterns that occur in the test words are bound to be contained in the training set, if we use disjoint sets of *words* for training and testing. Classifications of overlapping instances are bound to be correct, hence, it is advantageous to remember all instances.

- *modelling the relationship between frequency and regularity.* Regular correspondences tend to be more frequent in the training set than irregular ones. This counteracts the noise introduced by the irregular exemplars, because test instances are more often matched to regular exemplars than to irregular ones.

- *adequate similarity function.* Contrary to ANNs and decision trees, similarity functions can be manipulated and adapted very easily; the information–gain weighting is adequate for the task at hand.

SCT clearly suffers from the lack of data. Instead of checking feature values in a fixed order, it attempts to induce adequate tests from the data. For this, much data is needed if the relevant patterns are complex, as is the case with APT.

4 Conclusion

The results for Scottish Gaelic show that for minority languages, ML algorithms for APT may well be valid alternatives to devising rule sets by hand. The generalisation results of the best algorithms are tolerable for Scottish Gaelic, although it still remains to be seen if the frequency of errors seriously impedes intelligibility. Scottish Gaelic is a hard test case since its orthography is complex. For most small languages like Native American or African languages, orthographies were only devised in the last century and involve rather simple letter–to–phone correspondences. Therefore, for most other minority languages, the results should be even better.

Why build a ML–based module instead of a hand–crafted rule set? The main advantage of ML is that the difficulties in the phonetician's task are shifted from the acquisition and encoding of *knowledge* about a language to the encoding of *data*. Standard procedures exist for the latter which have been used by many fieldworkers, whereas the former may prove difficult, especially for languages with a complicated morpho–phonology. The basic lexicon for the TTS system can be used for training the APT module. Moreover, in building the lexicon, the user also creates a valuable resource for the further study of the language she works on.

IBL–based algorithms provide a particularly good interface to a TTS lexicon, since they provide a means of both *accessing* and *generalising over* the data stored there. This eliminates the need for a separate module for the transcription of unknown words.

Acknowledgements

The SCT software was kindly provided by the Institute for Computer Science V, University of Erlangen. The Stuttgart Neural Network Simulator (University of Stuttgart) was used for ANN simulations and LVQ_PAK (Helsinki University of Technology) for LVQ. M.W. would like to thank the Studienstiftung des Deutschen Volkes for funding.

References

Aha, D., Kibler, D., and Albert, M. (1991). Instance–based learning algorithms. *Machine Learning*, 6:37–66.

Allen, J., Hunnicutt, S., and Klatt, D. (1987). *From Text to Speech: the MITalk system.* MITPress, Cambridge, Mass.

Daelemans, W. and Van den Bosch, A. (1992). Generalisation performance of backpropagation learning on a syllabification task. In Drossaers, M. F. J. and Nijholt, A., editors, *TWLT3: Connectionism and Natural Language Processing*, pages 27–37, Enschede. Twente University.

Daelemans, W. and Van den Bosch, A. (1997). Language-independent data-oriented grapheme-to-phoneme conversion. In Van Santen, J. P. H., Sproat, R. W., Olive, J. P., and Hirschberg, J., editors, *Progress in Speech Processing*, pages 77–89. Berlin: Springer-Verlag.

Daelemans, W., Van den Bosch, A., and Weijters, A. (1997). IGTree: using trees for classification in lazy learning algorithms. *AI Review.* to be published.

Devijver, P. A. and Kittler, J. (1982). *Pattern Recognition. A Statistical Approach.* Prentice-Hall, London, UK.

Dedina, M. and Nusbaum, H. (1991). Pronounce: a program for pronunciation by analogy. *Computer Speech and Language*, 5:55–64.

Dietterich, T. and Bakiri, G. (1995). Solving multi–class problems using error–correcting codes. *JAIR*, 2:263–286.

Dietterich, T., Hild, H., and Bakiri, G. (1995). A comparision of ID3 and backpropagation for English text–to–speech mapping. *Machine Learning*, 18:51–80.

Gelfand, S., Ravishankar, C., and Delp, E. (1991). An iterative growing and pruning algorithm for classifier design. *IEEE Trans. PAMI*, pages 163–174.

Glushko, J. (1979). The organization and activation of orthographic knowledge. *J. Experimental Psychology: Human perception and performance*, pages 674–691.

Halle, M. (1992). Phonetic features. In Bright, W., editor, *International Encyclopedia of Linguistics*, pages 207–212. Oxford University Press, Oxford.

Kohonen, T., Kangas, J., Laaksonen, J., and Torkkola, K. (1996). LVQ-PAK - the Learning Vector Quantization package v. 3.0. Technical Report A30, Helsinki University of Technology.

Kolodner, J. (1993). *Case–Based Reasoning.* San Mateo, CA: Morgan Kaufmann.

Kuhn, R. and De Mori, R. (1995). The application of semantic classification trees to natural language understanding. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17:449–460.

Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming.* Chichester, UK: Ellis Horwood.

Mast, M., Niemann, H., Nöth, E., and Schukat-Talamazzini, E. (1995). Automatic classification fo dialog acts with semantic classification trees and polygrams. In *IJCAI, Workshop on "New Approaches to Learning for Natural Language Processing"*, pages 71–78, Montreal.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. In Rumelhart, D. and McClelland, J., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, Cambridge, MA.

Sejnowski, T. and Rosenberg, C. (1987). A parallel network that learns to pronounce English text. *Complex Systems*, 1:145–168.

Van den Bosch, A. and Daelemans, W. (1993). Data-oriented methods for grapheme-to-phoneme conversion. In *Proceedings of the 6th Conference of the EACL*, pages 45–53.

Weiss, S. and Kulikowski, C. (1991). *Computer Systems That Learn*. San Mateo, CA: Morgan Kaufmann.

Wolters, M. (1996). A dual–route neural–network based approach to grapheme–to–phoneme conversion. In v. Seelen, W., v.d. Malsburg, C., Sendhoff, B., and J., editors, *Proc. Intl. Conf. on Artificial Neural Networks 1996*, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, New York.

Wolters, M. (1997). A Diphone–Based Text–to–Speech System for Scottish Gaelic. Master's thesis, Department of Computer Science, University of Bonn.

Yvon, F. (1996). *Prononciation par analogie*. PhD thesis, École Nationale Supérieure des Télécommunications, Paris.